TASK: UA48
CDRL: 04107A
25 August 1992

Technical Concept
Command Center Library

Informal Technical Data

STARS-AC-04107A/001/00
25 August 1992

TASK: UA48
CDRL: 04107A
25 August 1992

INFORMAL TECHNICAL REPORT

For The

SOFTWARE TECHNOLOGY FOR ADAPTABLE, RELIABLE SYSTEMS (STARS)

Technical Concept
Command Center Library
Central Archive for Reusable Defense Software (CARDS)

STARS-AC-04107A/001/00
25 August 1992

Data Type: A005, Informal Technical Data

CONTRACT NO. F19628-88-D-0031
Delivery Order 0009

Prepared for:

Electronic Systems Center
Air Force Systems Command, USAF
Hanscom AFB, MA 01731-5000

Prepared by:

DSD Laboratories
under contract to
Paramax Systems Corporation
12010 Sunrise Valley Drive
Reston, VA 22091

TASK: UA48
CDRL: 04107A
25 August 1992

Data ID: STARS-AC-04107A/001/00

Distribution Statement "A"
per DoD Directive 5230.24
Authorized for public release; Distribution is unlimited.

Developed by: DSD Laboratories under contract to
Paramax Systems Corporation

This document, developed under the Software Technology for Adaptable, Reliable Systems (STARS) program, is approved for release under Distribution "A" of the Scientific and Technical Information Program Classification Scheme (DoD Directive 5230.24) unless otherwise indicated. Sponsored by the U.S. Defense Advanced Research Projects Agency (DARPA) under contract F19628-88-D-0031, the STARS program is supported by the military services, SEI, and MITRE, with the U.S. Air Force as the executive contracting agent.

TASK: UA48
CDRL: 04107A
25 August 1992

INFORMAL TECHNICAL REPORT
Technical Concept Document
Central Archive for Reusable Defense Software
(CARDS)

Principal Author(s):

Rose Marie Armstrong *Date*

W. Gregory Stine *Date*

Kurt Wallnau *Date*

Approvals:

Program Manager *Lorraine Martin* *Date*

(Signatures on File)

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	25 August 1992	Informal Technical Report

4. TITLE AND SUBTITLE
Technical Concept Command Center Library

5. FUNDING NUMBERS
F-19628-88-D-0031

6. AUTHOR(S)
DSD Laboratories

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Paramax Corporation
12010 Sunrise Valley Drive
Reston, VA 22091

8. PERFORMING ORGANIZATION REPORT NUMBER
STARS-AC-04107A/001/00

9. SPONSORING MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Department of the Air Force
Headquarters Electronic Systems Center
Hanscom AFB, MA 01731-5000

10. SPONSORING MONITORING AGENCY REPORT NUMBER
04107A

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION AVAILABILITY STATEMENT
Distribution "A"

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

The purpose of this document is to describe the technical concepts of the *command center* (CC) *library*, including library modelling, the library software *infrastructure*, security, and *interoperability*. This document will baseline the technical foundation for the CC library and for other *domain-specific libraries* to be implemented by the Central Archive for Reusable Defense Software (CARDS) program.

This document supersedes the Technical Concept Document dated 15 July 1991, and reflects the most current concepts being employed by CARDS. The next periodic update to this document should occur in February 1993.

14. SUBJECT TERMS	15. NUMBER OF PAGES
	42
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
unclassified	unclassified	unclassified	SAR

Table of Contents

List of Figures

1 Introduction

1.1 Purpose

The purpose of this document is to describe the technical concepts of the *command center* (CC) *library*, including library modelling, the library software *infrastructure*, security, and *interoperability*. This document will baseline the technical foundation for the CC library and for other *domain-specific libraries* to be implemented by the Central Archive for Reusable Defense Software (CARDS) program.

This document supersedes the Technical Concept Document dated 15 July 1991, and reflects the most current concepts being employed by CARDS. The next periodic update to this document should occur in February 1993.

Terms which appear in the glossary are italicized at their first occurrence in this document.

1.2 Background

There is a firm consensus that in order to increase productivity, quality and reliability, the software development community must *reuse* products from prior projects, as well as the associated human problem-solving expertise.

One of the tools used to encourage reuse is the *reuse library*. A variety of efforts, reflected in existing reuse libraries, have concentrated on the reuse of software code. This type of reuse has limited impact. The *reuser* often has to struggle to insert *components* into a system that is in the coding stage of the software development *life-cycle*.

Today, when components (i.e., a set of reusable resources) are inserted into current operational libraries, they are typically classified in broad, generalized categories, and information describing their specific role in a particular problem-domain architecture is not formally *encoded*. As a result, the *domain* knowledge is no longer attached to the component and therefore lost to the reuser.

To achieve broad spectrum reuse, CARDS advocates *library-centered domain-specific reuse*. A domain-specific library is built around a domain that has been carefully scoped and modeled. A domain-specific library for command centers (CC) was created under Phase I of the program. As the library matures, it will build towards having a full set of *life-cycle artifacts*, including both *domain engineering* and system/software engineering artifacts.

1.3 Scope

This document addresses the technical tasks specific to the continued development and maintenance of the CC library required during Phase II of the CARDS program as described in the UA48 Statement of Work dated 13 February 92, issued by HQ ESC/AVS. References made to future efforts beyond the scope of Phase II are included for informational purposes only.

1.4 Mission of CARDS

The CARDS program is a concerted DoD effort to transition advances in the techniques and technologies of domain-specific software reuse into mainstream DoD software procurements. There are three key elements to the CARDS approach:

1. Develop and transition, through a Franchise Plan, a "*knowledge blueprint*" for *domain-specific reuse* to the DoD and DoD software development industry, using the library as a tool.
2. Develop and transition a training plan, training courses, and adoption handbooks as vehicles for enhancing the penetration of the Franchise Plan and general reuse practices.
3. Apply domain-specific reuse techniques and technologies to produce an operational library for command centers.

The knowledge blueprint will be conveyed by a Franchise Plan consisting of library documentation, reuse process handbooks, cost, schedules, education and training documentation, and procedures for implementing domain-specific reuse. The Franchise Plan will support the transition of the blueprint to DoD organizations and contractors as an overall plan for developing and supporting other domain-specific infrastructures. The handbooks will be targeted to audiences (including direction-level and acquisition personnel, engineers, and tool and component vendors) crucial to the adoption and institutionalization of library-centered domain-specific reuse techniques. The handbooks will address the issues and responsibilities of the intended audience as it pertains to the implementation of domain-specific reuse.

A training plan and system/software engineers' course will be developed to support the integration of domain-specific reuse into the software development life-cycle and educate and reeducate software professionals and support the elimination of cultural barriers.

The establishment of a domain-specific library for the command center domain will continue in support of the CARDS mission for evaluating and validating the knowledge blueprint. The CC library will be maintained at a central site and will consist in part of: facilities, personnel, domain and library *modeling*, software components, qualification procedures, maintenance procedures, *retrieval* procedures, *browsing* capabilities, and various user services that support domain-specific reuse in this library. User interaction with the library will occur from multiple, remote sites.

CARDS is working closely with *PRISM* (Portable Reusable Integrated Software Modules) to develop the command center reuse library. PRISM will develop *generic architectures* and prototype implementations of command centers which CARDS will use as a basis for the domain-specific library.

Ultimately, CARDS will support the full software development life-cycle by providing mechanisms and methods that support domain-specific reuse integration, *component acquisition* policy, *rapid prototyping*, clarification of requirements, *system composition* and component generation.

1.5 Document structure

The four chapters of this document cover: (1) Introduction, (2) Reuse Library Infrastructure, (3) CC Library Support Infrastructure and (4) Library Interoperability. Chapter 2 (on the reuse library infrastructure) covers domain engineering and its relationship with *system engineering*, modeling concepts, operational concepts and tools, and the CARDS/PRISM (Portable Reusable Integrated Software Modules) relationship. The third chapter covers the technical aspects necessary to make the CC library operational. Chapter 4 (on library interoperability) covers key technical concepts of library component exchange.

2 Reuse Library Modeling Infrastructure

The CARDS approach to constructing and using reuse libraries differs from other approaches in a significant way; rather than viewing a library as merely a "*repository*" — a storage area for software components — CARDS views a library as a *library model* and a set of *library applications*. Thus, CARDS distinguishes the concept of a reuse repository (the underlying storage) from the reuse library (the storage and related applications).

On the surface this is a subtle distinction, since even repositories such as Asset Source for Software Engineering Technology (*ASSET*) and Defense Software Repository System (*DSRS*) require some underlying *model* (commonly referred to as "the *data model*"). At a deeper level however, this distinction is important in understanding the CARDS approach to library-centered domain-specific reuse. Some immediate implications of this distinction are:

- While conventional repositories are focused exclusively on *reusable components*, CARDS extends this focus to include the *relationships* that exist between components.
- The definition of component in conventional repositories tends to follow easily described, well-partitioned functional lines, e.g., documents, subroutines, modules, and applications. In CARDS, components are not always as discrete, and include concepts such as requirements, generic architectures and other conceptual models. Thus, CARDS functional components can be related to clusters of requirements or to domain-specific application architectures.
- Conventional repositories place a heavy emphasis on component search and retrieval, i.e., they are usually characterized by a single application supporting interactive search. The CARDS approach envisions a collection of library applications tailored to the domain of interest and a selected clientele. Sample applications include: *graphical browsers*, system composers and component qualifiers.

To achieve this technical vision of a reuse library, CARDS relies upon the use of modeling formalisms that are significantly richer than lower-level data modeling formalisms such as the entity-relationship-attribute (ERA) models and relational models which characterize conventional repository approaches. Instead, CARDS draws upon technology derived from the field of *knowledge representation*.

The use of knowledge representation techniques is justified as, and is crucial to, the means of describing, managing and using the complex sets of relationships (also referred to as constraints) that characterize an *application domain* and its *software architectures* and components. (A detailed description of CC library constraints is included later in this document.)

It is not enough, however, to have the tools available to create library models and library applications — there are critical design decisions which must be made in the design and construction of domain-specific libraries that depend upon:

- The nature of the underlying application domain

- The manner in which the underlying application domain is analyzed and modeled, i.e., *domain analysis* techniques
- The anticipated end-user requirements on the library, i.e., usage scenarios
- The capabilities of the modeling system used to create the library model

In addition to the above, there are technical issues related to the support and management of an operational library that transcend the engineering mechanics of library design and construction. The mechanics of library design and construction and the technical concepts of library operations must be considered as part of the same engineering process for creating and fielding an operational domain-specific reuse library.

The following sections describe the interplay of these (and other) dimensions of the library design and construction processes. Section 2.1 describes how the underlying application domain, domain analysis, and library end-user scenarios affect the design of a domain-specific reuse library and how the CARDS library infrastructure can be applied to design and implement domain-specific reuse libraries. Section 2.2 discusses how the issues described in Section 2.1 apply to the construction of the CC library. Chapter 3 discusses the technology issues of operational libraries, with emphasis on the CC library.

2.1 Library design and construction

Utilization of reuse in narrow application domains and the construction of domain-specific libraries is not new, e.g., statistical and mathematical libraries. However, the attempt to make the practice of library-centered domain-specific reuse more systematic and repeatable across different application domains is a recent phenomena, and has received heightened attention in recent years.

One result of recent efforts is the differentiation of two distinct engineering life-cycles: domain engineering and system engineering [4]. Domain engineering refers to the techniques (i.e., methodologies) used to analyze and model an application domain, and construct reusable components based upon these analyses and models. System engineering refers to the more familiar world of software and system development processes.

This differentiation conveys a number of important principles:

- Although the representation techniques of domain engineering are frequently used in system engineering (e.g., structured analysis and design technique (*SADT*) diagrams), their intent and meaning are different.
- To be successful, domain engineering activities must be considered independent of any single application.
- Domain engineering implies an investment approach to the software life-cycle, where the costs of instantiating a domain-engineering process will be amortized over several system-engineering instantiations.

- The above-noted economic factors can be viewed in this way: domain engineering can be considered a "supply-side" process, while system engineering is a "demand-side" process. To be successful, a library-centered domain-specific reuse strategy must balance the needs of the demand side with products produced by the supply side.

Elaborating on this last point, one element of a risk reduction strategy for amortizing the costs of domain engineering includes the techniques and technology of ensuring the availability and use of domain engineering by-products for system development processes. CARDS views this "availability and use" requirement in terms of integrating the domain and system engineering life-cycle processes and the domain-specific library as the underlying technology which supports this integration.

2.1.1 Library integration of supply and demand processes

Integration should be thought of in terms of relationships between two or more integrated entities [19]. In the context of this report, these two entities are life-cycle processes. Figure 2-1

Figure 2-1. Domain and System Engineering Processes

characterizes the domain and system engineering processes, and draws parallels between these processes [4]. While this characterization has obvious limitations (not the least of which is the lack of industry consensus on the precise meaning of the terms used to describe domain engineering in Figure 2-1), it does illustrate a number of important parallels which are important in understanding the integration of these processes:

- Domain requirements result from a requirements analysis for an entire family (i.e., domain) of applications; conversely, application requirements are targeted to a specific application.

- One result of domain analysis may be the specification of a *domain architecture* which is used to convey high-level implementation paradigms and constraints characterizing commonality and variances of domain applications; conversely, application architectures are focused on satisfying a particular set of application requirements.
- Another result of domain analysis may be domain implementations, i.e., systems, subsystems and components which implement, or support the implementation of, the domain architecture; conversely, application implementations are targeted to a specific application architecture.

Given these parallels, there are many ways of making use of the products of domain engineering during system engineering, i.e., integrating these activities. Three possibilities are illustrated in Figures 2-2a through 2-2c.

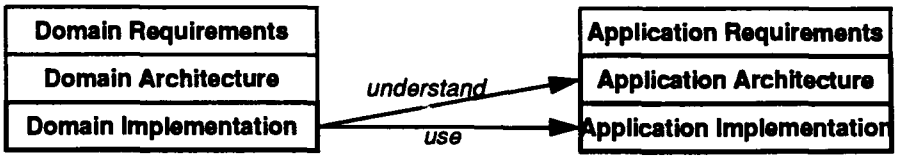

Figure 2-2a. Implementation-Level Integration

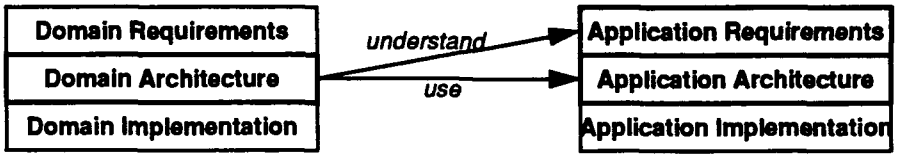

Figure 2-2b. Architecture-Level Integration

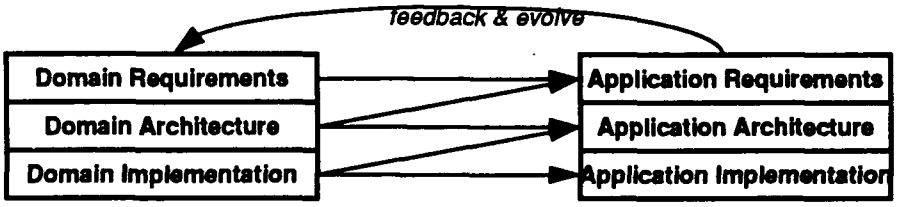

Figure 2-2c. Domain-Level Integration

Figure 2-2. Integrating Domain and System Engineering

Figure 2-2a illustrates two ways the domain implementation-level components can be used during the system engineering process:

- During "bottom-up" design of an application architecture, the system designer can integrate, and understand, the domain implementations;
- As part of the application implementation, the software engineer can locate, and use, the domain implementations.

Figure 2-2b illustrates a similar understand/use dichotomy, but at a higher level of abstraction in the system engineering process. In this case:

- An understanding of a domain architecture can aid a requirements analyst in analyzing and allocating requirements;
- A system designer can make direct use of the domain architecture in developing an appropriate application architecture.

Finally, Figure 2-2c illustrates the scenario where a more complete set of products from domain analysis are used through a broader spectrum of system engineering activities, and where the results of successive instantiations of system engineering life-cycles feed back to the domain products to incorporate application-specific variations. Figure 2-2c can be considered a model of the ideal integration of domain and system engineering.

While Figure 2-2c may be an ideal scenario, a number of technical and economic factors may constrain the nature of the endpoints of the integration relationships. Examples of such factors include:

- The underlying domain may not be structured in a way that is meaningfully conveyed as a domain architecture, e.g., domains of missile guidance algorithms, mathematical routines, digital sound samplings, etc. This illustrates scenarios where domain-products of *horizontal domains* or non-architectural *vertical domains* are integrated with applications.
- The application domain may not be sufficiently stable (in terms of requirements, architectures or implementation technology) to warrant the investment of developing, or to expect the existence of, domain implementations. This illustrates scenarios where domain analysis and architecture specification is occurring simultaneously with application development, perhaps as a means of *prototyping* domain implementations, as is the case with Portable Reusable Integrated Software Modules (PRISM)/Generic Command Center (GCC) developments. (See section 2.2.1.)
- The analysis techniques used to produce domain products may not have generated all of the products depicted in Figures 2-1 and 2-2, or may not have generated products useful to a particular system engineering process. This illustrates scenarios where the domain engineering life-cycle may have been constrained by economic factors to generate only a partial set of products (e.g., requirements but no architecture), and scenarios where different organizations with incompatible notions of supply and demand-side processes produce or require different domain products.

- The system engineering processes may not be able to make use of domain products for reasons of process maturity. For example, an organization struggling to codify and institute repeatable analysis and design processes may find it more practical to attempt a modest level of supply-side/demand-side integration, such as *implementation-level integration* as illustrated in Figure 2-2a.
- The underlying means of integrating domain engineering with system engineering — the reuse library — may not adequately model, or provide access to, domain requirements or architectures. This illustrates scenarios derived from the use of conventional component-oriented reuse repositories.

The above factors are by no means exhaustive, but are intended to illustrate the following point:

> *The manner in which supply-side and demand-side reuse processes are integrated depends upon many factors, including the nature of the domain, the kinds of domain-oriented components that are available, the nature of the supply-side processes, and the capabilities of the reuse technology used to convey the results of domain engineering to system engineering processes.*

All of these points are of particular importance to the CARDS program because CARDS must attempt to convey in its reuse blueprint the techniques and technology necessary to develop domain-specific reuse libraries in a way that is independent of:

- application domain
- domain analysis methods
- system engineering methods

Thus, CARDS is attempting to describe the means of integrating demand and supply-side processes without constraining the nature of either of these processes. For this reason, CARDS finds it necessary, and convenient, to view the creation of a domain-specific library as an activity which is independent of domain and system engineering life-cycles.

2.1.2 Domain vs. library modeling

Part of distinguishing domain engineering from library engineering is differentiating domain analysis and modeling from library analysis and modeling. Domain analysis refers to the analytical processes for scoping domains, identifying the key concepts, common and variant features, etc. *Domain modeling* refers to the formal characterization of the results of analysis in some representational form (ERA models, feature models, taxonomies, SADT diagrams, glossaries, etc.).

Library analysis, on the other hand, refers to the analytical processes for constructing the library system used to integrate the results of domain engineering into system engineering processes. As already mentioned, this design activity needs to take into consideration many factors affecting this integration relationship. Library modeling refers to the formal characterization of the library system. In some domain analysis techniques, some (but not all) aspects of the library modeling are produced as domain analysis by-products, e.g., facets from Prieto-Diaz's methods [17].

One way to view the separation of domain modeling from library modeling is depicted in Figure 2-3. The *domain model* encompasses things such as glossaries, *context models* and economic models, while the library model encompasses search heuristics and prototyping support services, in addition to meta-level information about various components in the library (with *COTS* license data being the illustration of this in Figure 2-3). Both models overlap in their use of domain requirements and *architecture models*. Note that this figure is an illustration drawn from the CC library; different library instantiations may choose different partitions.

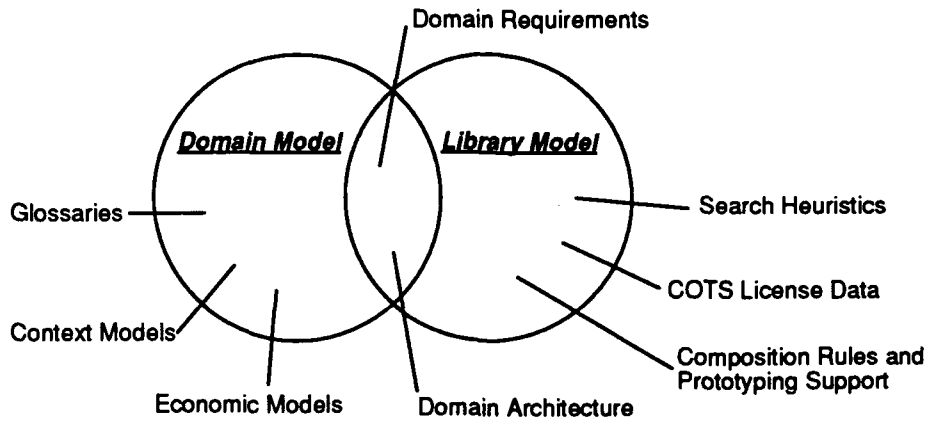

Figure 2-3. Relationships between Library and Domain Models

The central issue raised by the separation of library analysis and modeling from domain analysis and modeling is harmonizing the content of the library model with that of the various models produced during domain analysis. Various questions need to be answered, including:

- Which models produced by domain analysis should be used to produce a library model?
- Should domain models be "mapped" into a library modeling formalism, or should the domain models remain in their original formalism? (And if so, how should this formalism be integrated into the library model?)
- How should the library model be kept consistent with the domain model, in cases where the underlying domain and/or domain analysis continues to evolve?

Answers to questions such as these depend upon the nature of the domain and domain analysis processes, the nature of the library modeling system and formalism used, and the kinds of applications to be developed for the library. Before describing how these questions are being answered for the CC library, a discussion of domain and library modeling, and the CARDS reuse library infrastructure, is in order.

2.1.3 Domain modeling

The techniques used to model domains are as numerous as proposed domain analysis methods. Domain modeling formalisms include:

- feature lattices [6]
- SADT diagrams [16]
- *ERA* diagrams [6]
- domain-specific languages [15]
- module-interconnection languages [12]
- *semantic networks* [9]

to name just a few. One of the justifications for separating library modeling from domain modeling is precisely this diversity in domain modeling formalisms — and no industry consensus on domain modeling formalisms appears to be emerging.

Despite this diversity, however, some key concepts about domain analysis and modeling have emerged, specifically as these analysis and modeling techniques relate to both domain-specific reuse and model-based engineering [10]. These key concepts include:

- Specification of a domain architecture, i.e., a model conveying a high-level specification of implementation paradigms for applications within a particular domain
- Separation of the domain architecture from specific realizations of the architecture, and from the requirements of applications within the domain (as implied in Figures 2-1 and 2-2)
- Capture of both commonality and variation within the domain.

CARDS considers these to be the key concepts of domain analysis because they have direct bearing on library modeling. The meaning and use of domain architectures is discussed below to provide a foundation for understanding the CARDS library modeling approach taken for the CC library.

2.1.3.1 Domain architectures

Applications in a particular problem domain tend to exhibit similarities in components and in the architecture in terms of the arrangement and use of components. Experts in a particular domain usually have an informal, often unwritten, model in mind when constructing a system. An architecture provides a formalized version of such a conceptual model.

While a precise and universally accepted definition of the term "software architecture" remains a topic of philosophical debate, CARDS use of the term is based upon the understanding that the domain architecture:

- defines the functionality of, and interfaces between, major subsystems of applications within the domain
- provides the basis for constructing and relating domain implementation components
- provides the basis for mapping (or allocating) domain requirements to domain components, and to specific implementations of the domain architecture created by system engineering processes.

There are, of course, other uses for domain architectures. For example, a domain architecture can serve as the basis for creating industry-wide standards for applications within a particular domain. However, this report is focused on the application of domain architectures to library modeling.

2.1.3.2 Domain architectures: genericity versus abstraction

The above definition of domain architecture is based upon an interpretation of the purpose of a domain-specific software architecture which is, unfortunately, not widely understood or universally appreciated. That is, CARDS believes the purpose of a domain architecture is to act as an abstraction which can be used to describe many different implementations, and can satisfy different requirements.

This may seem like a reasonable interpretation of the purpose of a domain architecture. There is, however, an alternative view, one which CARDS believes is less flexible: that a domain architecture describes only the commonality among different applications in the underlying domain. This view of domain architecture is often conveyed by terms such as "generic architecture" (although not every use of the term generic architecture necessarily conveys this meaning).

To distinguish CARDS use of the concept of domain-specific software architecture from this less flexible view, the term domain architecture will be used in preference to generic architecture.

2.1.3.3 Capturing commonality and variation

An argument for commonality

On the commonality side, an architecture which defines stable interfaces among subystems and components provides the basis for standardization. When well-defined standard interfaces are present, one is free to select a component based upon its ability to meet mission requirements, without having to be concerned with low-level interface issues. Standard interfaces enable construction of components with newer, high-quality algorithms, resulting in increased accuracy, performance, and other improvements without the need for major restructuring of previously developed components or full-scale applications. This enables an orderly evolution to increased capabilities, performance, and quality.

In the long run, commonality can also form a basis for wide-scale industry and Government agreement on the specifications for reusable components for a particular domain. This would

provide the kind of specification stability required to support the birth of a software component industry.

An argument for variation

Requirements for individual command centers vary in subtle ways, such as variations in capacity, performance, use of real-time data, cost and so on. It is unrealistic to expect that for all domains any specific implementation of a domain-specific software architecture can satisfy a sufficiently broad set of domain requirements to allow amortization of the domain engineering investments over sufficiently many system engineering life-cycles. More flexibility is required to allow selection of some subsystem implementations and the flexible composition of variations of other subsystems from lower-level domain components.

An argument for commonality and variation

Architectures capturing both commonality and variation are important for the construction of domain-specific reuse libraries. With a domain architecture describing commonality and variation, and a rich supply of ready-to-use parts which (alone or in combination) can satisfy the architecture, prototypes for new applications can be quickly and easily constructed from unique combinations of preexisting components. Once a prototype closely matches the desired objective, requirements and implementations for the operational system can then be extracted, providing the basis for construction of an operational system.

This ability to quickly develop prototypes to satisfy, and help specify, requirements for a new system is fundamental to the CARDS approach and is critical to the success of the CARDS mission.

2.1.4 Library modeling

As noted earlier in this document, CARDS views a library as a library model and a set of applications, and the construction of a library model as a design activity which balances various requirements. What "goes into" and what "comes out of" the model is dependent upon many factors, including: the library modeling formalism used, characteristics of the domain engineering life-cycle (e.g., the kind of domain analysis that was conducted) and characteristics of the system engineering life-cycle (e.g., the kind of library applications that need to be constructed to support the anticipated system-engineering processes). Section 2.2 provides details on how some of these issues are addressed for the construction of the CC library.

The main point to note about library modeling is that the library model includes information in addition to that derived from, or pertinent to, domain analysis. This point is illustrated in Figure 2-3. Information supporting the supply-side (i.e., user demands) can include:

- meta-level *attributes* describing elements that are derived from the domain model, such as commentary or other documentation on domain requirements, architectures and components

- information used specifically to support one or more library applications, such as graphical browsers, system composers and component qualifiers
- inter-library information, such as indexes to other library models (from other domains, i.e., model intraoperation) and indexes into other library systems (i.e., interoperation)
- integration of other representation schemes, possibly through the invocation of CASE design tools.

2.2 CC library design and construction

The initial CARDS CC library is using the results of the Portable Reusable Integrated Software Modules (PRISM) program. The CARDS and PRISM programs have evolved into a cooperative working arrangement whereby PRISM acts as the source of and provides expertise in command center technology, while CARDS acts as the source of and provides expertise in library technology. This cooperative working arrangement evolved in part from the PRISM selection of CARDS as the PRISM repository. (The term repository is used intentionally, as this was the term used by PRISM). This selection and the process leading to it, are described in the PRISM repository selection report [18].

2.2.1 PRISM program and approach

The PRISM program is engaged in simultaneous domain engineering and system engineering life-cycles. The PRISM domain engineering work is performing a domain analysis, and is engaged in an effort to define a generic command center (GCC) architecture. This definition is proceeding in parallel efforts that incorporate both top-down and bottom-up design.

The top-down analysis is driven by in-house expertise in command center requirements and past implementations; and by examination of existing command center systems and documentation, such as the Defense Information Systems Agency (*DISA*) Command Center Design Handbook [11]. Simultaneously, a command center prototyping effort is refining and expanding the generic architecture to accommodate increasing functionality and "lessons-learned" from prototyping efforts. An additional aspect of the PRISM approach is a heavy emphasis on the reuse of existing government off-the-shelf (*GOTS*) and COTS components to implement the GCC. The dual action of analysis and prototyping, and the focus on COTS and GOTS components, is a reasonable approach to grounding theoretical domain analysis in a practical setting.

One goal of the PRISM program is to establish "the command and control (C^2) store" — a repository of command center components — which can be used in conjunction with the GCC and prototypes to reduce the time required to field an operational command center. The PRISM stated goal is to support prototyping of approximately eighty percent (80%) of a functional command center with the PRISM architecture and C^2 store.

The PRISM approach and objective has a number of consequences on CARDS library modeling:

- the PRISM GCC is evolving at a rapid pace, as is the functional capability of the GCC prototype
- the by-products of the PRISM domain analysis are tightly coupled with the GCC prototypes
- the CARDS CC library needs to support prototyping activities.

Each of these objectives has an impact on the design and construction of the library model and applications comprising the CC library. The following sections expand upon these concepts.

2.2.2 CC library overview: the model and its applications

The CC library consists of a library model derived from documentation and prototypes produced by the PRISM program, and from existing and planned applications for CC library users. These applications consist of:

- graphical browser (already developed as part of the reuse library infrastructure)
- system composition tool (in prototype development)
- *component qualification* tool (planned for development)

The PRISM program provides the core of the CC library model in the form of documents describing command center requirements, a *generic command center architecture* (the GCC architecture) and prototype command center implementations. These documents and prototypes are then analyzed and encoded into a library model using the *STARS* (Software Technology for Adaptable, Reliable Systems) Reuse Library Framework (*RLF*).

The library modeling services provided by the RLF [1, 2] include a structured inheritance network formalism similar to KL-ONE [5] and a specialized rule-based inferencing system. Library application development is supported by an open systems specification to the structured inheritance and rule-base systems. Section 2.2.3 describes, in more detail, the approach taken to using the RLF modeling formalism to model the command center domain, while section 2.2.4 describes, in some detail, the architecture and computational model of the system composition application.

2.2.3 CC library modeling approach

The approach to modeling the CC library is evolving to support the system composition application (in addition to the browsing application). It is not surprising that this new application results in a refined view of how the CC library model should be structured, and what kind of information is needed in the model to support automatic system composition (section 2.2.4). Figure 2-4 illustrates this new approach.

The revised modeling approach attempts to partition the library model (Figure 2-4) into three major groupings of constraints: domain constraints, architectural constraints and implementation constraints. These groupings correspond to the products of domain engineering described in

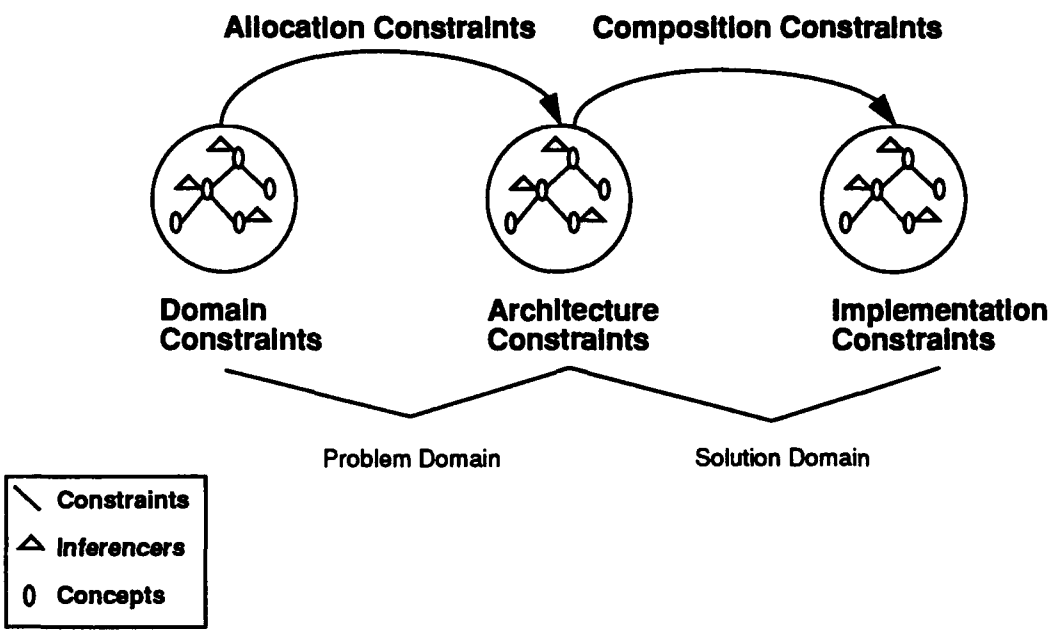

Figure 2-4. Command Center Library Sub-Models

section 2.1.1, and produced by the PRISM program. At a high level, the purpose of this partitioning scheme is to allow a differentiation in the modeling of the problem space, traditionally the purview of domain engineering, from the solution space, the purview of system engineering. CARDS believes that this partitioning will:

- help library modelers focus on capturing the right kinds of information to support system composition (and future applications)
- isolate portions of the model which are likely to undergo evolution (particularly the implementation and architecture constraints)
- better support library users by allowing users to work at several levels of abstraction:
  - mission-level abstraction, such as "the system must support global monitoring of these events"
  - architectural-level abstraction, such as "I am interested in the message processing subsystem of a command center"
  - implementation-level abstraction, such as "I want to use Ingres, not Sybase, for my command center".

The domain constraints attempt to capture the key requirements supported by command centers that can be composed via the CARDS CC library. The architecture constraints attempt to capture the high-level model of subsystem relationships and their key functionality (as expressed in PRISM documentation). Finally, the implementation constraints capture the lower-level constraints that describe the relationships of particular COTS and GOTS components with each other and with the underlying computing platform.

Between these groupings there are other constraints that need to be mapped between the various sub-models. Between the domain and architecture constraints there are allocation constraints. These constraints model the allocation of requirements to the architecture; this mapping is, for the most part, described in PRISM documentation, and supports the traceability of key requirements to parts of composed systems. Between the architecture and implementation constraints there are composition constraints. These constraints show which components, and any related components, are used to implement some part of the architecture and support the system composition tool in composing systems that are consistent with the domain architecture.

2.2.4 System composition application

The objective of system composition is to provide the CC library users with tools to automate the composition of new command centers, or portions of command centers, from components within the CC library. The approach is to apply user-provided constraints to the CC library model to produce prototype demonstrations of systems, assist users in the decision making process of building new systems, and provide users with the actual software to build them (when possible).

Figure 2-5 provides a top level view of the system composition application. There are three inputs

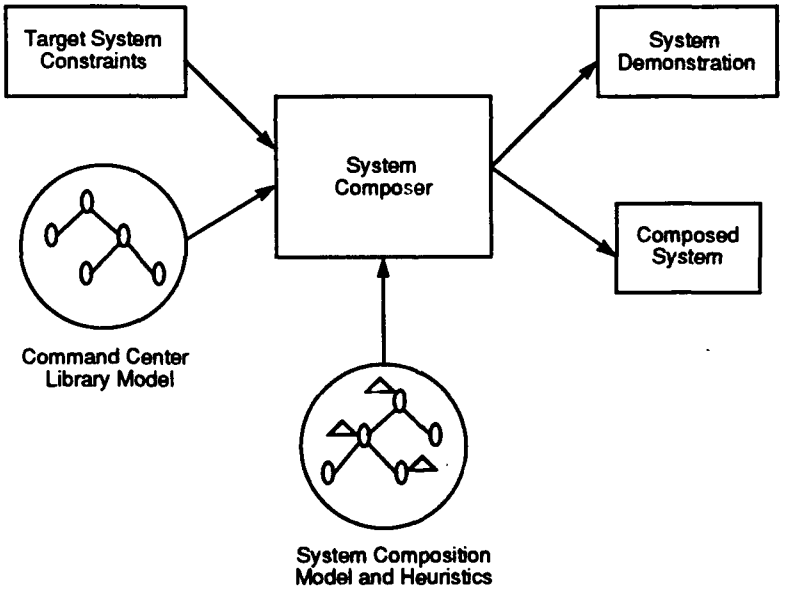

Figure 2-5. System Composition - Top-Level Architecture

to the system composition application (called "System Composer" in Figure 2-5): a model of the command center library, target system constraints elicited from the user through the user interface, and a model of system composition and heuristics for building the system. The outputs of the system composition tool are system demonstrations and composed systems (or portions of a system).

2.2.4.1 Target System Constraints

The target system constraints are acquired from the user through a structured dialogue. This dialogue is controlled, and defined, by both the structure of the command center library model, and by the system composition model. These constraints convey:

- requirements on system functionality, such as the kinds of functions, tasks and activities the command center will support [11];
- aspects of the domain architecture of interest to the user, such as message processing and geographic information systems;
- implementation constraints, such as platform constraints (e.g., Sun vs. HP), system software constraints (e.g, X11/Motif vs. OpenWindows), COTS constraints (Ingres vs. Oracle), or performance constraints (1500 message per second), etc.

2.2.4.2 System Composition Model and Heuristics

The system composition model is "consulted" by the system composer to determine when and how to query the user for the necessary user-defined requirements. The system composer transverses the CC library model and at various concepts in the model, the system composer may have alternative strategies for composing a system. If the *concept* has several *specializations*, the composer may need to elicit information from the user to help determine which of the specializations is most appropriate; conversely, the composer may have default rules to consult, either built-in to the system composition model, or encoded as part of the CC library model.

2.2.4.3 System Demonstration

Part of the process of determining which variants of a domain architecture, or which implementation of this architecture, are of interest to the user is the demonstration of composed system functionality. Although in practice there will be some restrictions on which variants of the system can be demonstrated (for reasons of license restrictions, the degree to which the components are integrated with the CC library model and with each other, etc.), the goal of the composition tool is to support iterative, real-time demonstration of system variants.

2.2.4.4 Composed System

At some point, the CC library user will want to "extract" from the system the results of the composition. The "Composed System" box in Figure 2-5 represents the types of information the user may be provided with, in a form suitable for extraction and conveyance to the user. Examples of such information include:

- the software "wrappers" that provide integration among two or more COTS/GOTS components
- vendor and licensing information about COTS components used in the composition

- implementation information about the composed system, including performance information, hosting constraints and dependencies, module interconnection information, etc.
- documentation about the composition, such as the rationale used by the system composer in addition to a log of the user interaction with the composition tool.

2.2.4.5 Current status of the system composition tool

The system composition model is designed to be domain and application independent; however, the degree to which this independence can be achieved has not yet been demonstrated. If independence is possible, the system composition model can be replaced, e.g., by a qualification model to produce an application which automatically generates component qualification plans. This goal is important since it supports reuse of library application models in different library domains.

Currently, the system composition tool exists in a prototype implementation completely utilizing the GOTS rule-base inferencing system, C Language Integrated Production System (*CLIPS*). This prototype is being transitioned to a more complete implementation making use of the RLF modeling formalism for the semantic net structure (as described in Section 2.2.3) and CLIPS for the rule-based inferencing.

The existing prototype focuses primarily on the message processing systems of the Phase III PRISM prototype. The following components are presently integrated into the system composition process:

- message generator (text and graphical)
- system manager (with and without MTV)
- Message Translation and Validation (MTV) providing vanilla SQL, Sybase and Ingres DBMS)
- databases (Ingres/Sybase)

Several flavors of the message processing subsystem can be automatically composed and executed, based on the above components.

2.2.5 Component qualification

Components for use in command center applications may be obtained from many sources. Identification of new components for inclusion in the CC library will be accomplished in conjunction with input from *domain experts*. In the near-term, the principal source of information regarding candidates for inclusion into the CC library will be the PRISM program. In the longer-term, components from other libraries, DoD software programs, other Government agency programs, and COTS will be added as they are identified and acquired. Selection of these additional components for incorporation into CARDS will be based on their compatibility with the CC library model, as well as other acceptance criteria common across multiple domain/library models.

Component qualification involves a detailed assessment of a component with respect to a specific set of requirements. These requirements include a set of criteria common to each component and criteria resulting from the specific requirements determined by the domain, architecture, and implementation constraints. The common criteria are derived from the PRISM Qualification Methodology[14] and are intended to measure the degree of reusability for the component using factors such as portability, reliability, cost effectiveness, performance, etc. The *domain criteria* measure the component's "fit, form, and function" against the constraints imposed by the requirements, architecture, and implementation. Additionally, the results of the domain criteria may also serve to interpret the analysis of the component against the common criteria. Developing and applying metrics measurement of a component's conformance to domain criteria would most likely be an iterative process resulting from an initial lack of formality and completeness of specifications. Each cycle within the evaluation process involves further refinement of the domain criteria with consultation of the domain expert.

Both CARDS and PRISM will share a common qualification methodology that provides the detailed assessment of a product with respect to a specific set of component requirements along with a general set of selection criteria common to each component in the CC architecture. CARDS will attempt to utilize, whenever possible, the selection criteria when developing a library for other domains. It is possible that criteria exist that are domain-specific and therefore, for each domain, some criteria will need to be developed.

3 CC Library Support Infrastructure

In addition to the modeling formalisms presented in chapter 2, there are other technical considerations for building and maintaining a domain-specific library. The support hardware, software, network and telecommunications installed in the CC library facility provide the base upon which the library system is built. The supporting base consists of Sun workstations, the Sun operating system (SunOS), Internet and Ethernet, various compilers, configuration management tools, RLF, *AFS*, and the X Window System and associated window managers. The underlying system software, specifically the X Window System and AFS software, supports access to the central library from the remote and central sites. The various portions of the library infrastructure are detailed below.

3.1 The CARDS library infrastructure

Figure 3-1 illustrates the software infrastructure used for creating domain-specific libraries during Phases I and II of the CARDS program. The Base RLF Services are described in the *AdaKNET* and *AdaTAU* user's manuals [1, 2]. AFS is described in the AFS System Administrator's Guide [3], while the use of AFS and X in support of a distributed CARDS architecture is discussed in Section 3.2. The component qualification methodology was discussed in section 2.2.5 and section 2.2 described the key concepts of the CC library model and its applications.

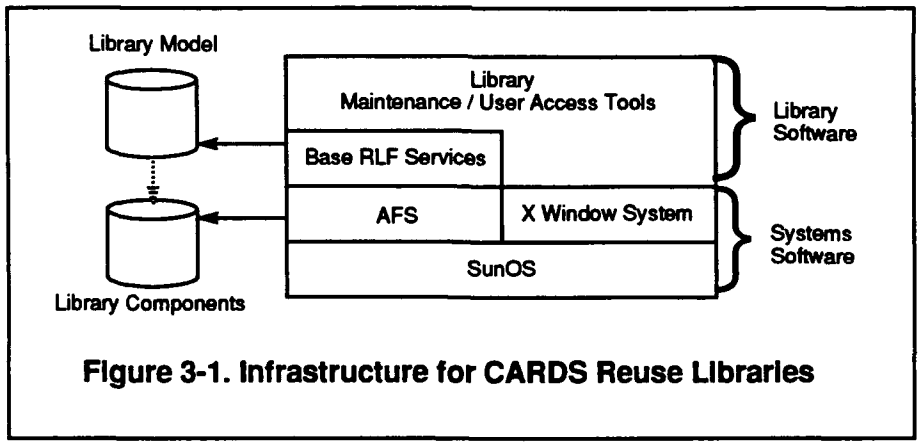

Figure 3-1. Infrastructure for CARDS Reuse Libraries

3.2 Distribution

The library software architecture supports three different forms of system distribution (complete via AFS, partial via AFS, or via the X Window System) as well as undistributed use at the central site. The CC library WAN architecture is shown in Figure 3-2.

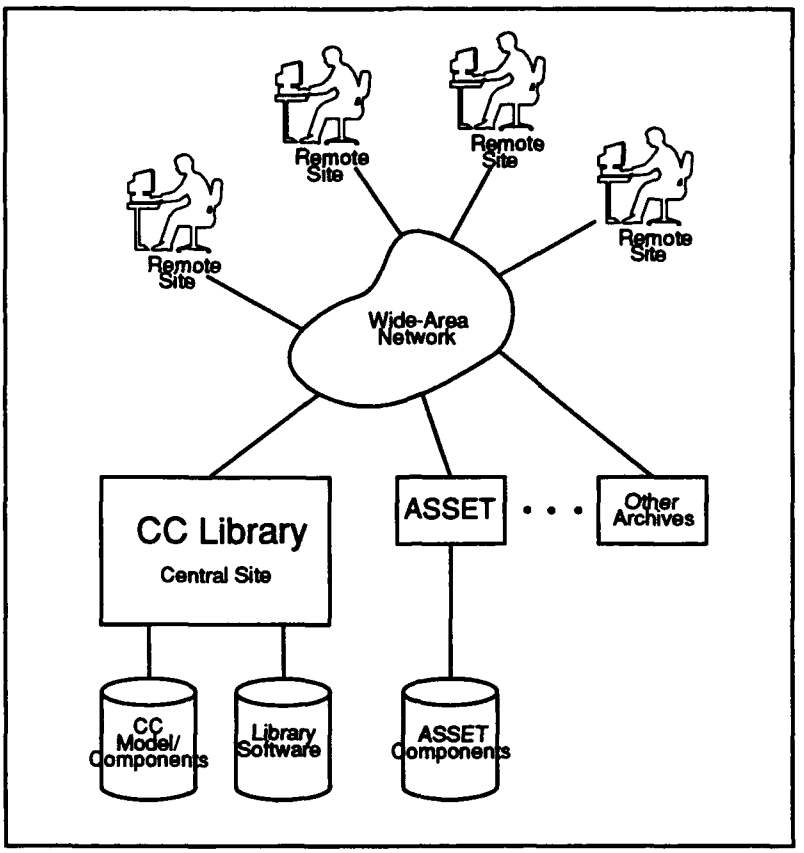

Figure 3-2. CC Library Wide-Area-Network Architecture

Figure 3-3 shows how the X Window System, AFS and RLF all reside on the central system. Through use of this architecture, and with the addition of a wide-area network, remote access sites can be configured to interconnect with the central site. These options and their advantages and disadvantages are described below.

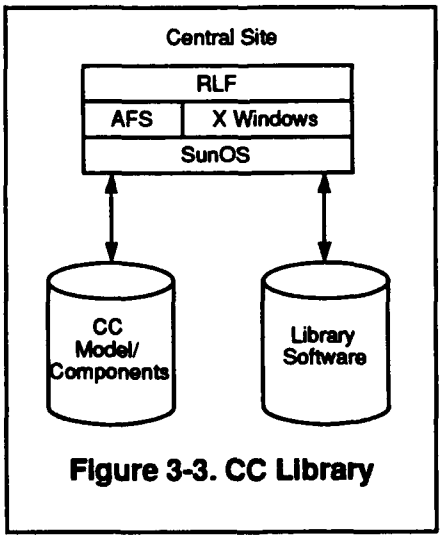

Figure 3-3. CC Library

3.2.1 Complete distribution via AFS

The primary method of distribution for the CC library is via AFS (Figure 3-4). AFS provides direct access to the central site's file system from the remote sites, user *authentication* options and protection for the library files which will either allow or prevent unauthorized access to the system. AFS provides a caching facility that allows software and files to reside on a remote system and operate on a local workstation across a wide-area network. The software and files accessed by a user are cached to the user's workstation on the initial access only. Additional accesses of the same file or software will be from the cached space on the local workstation.

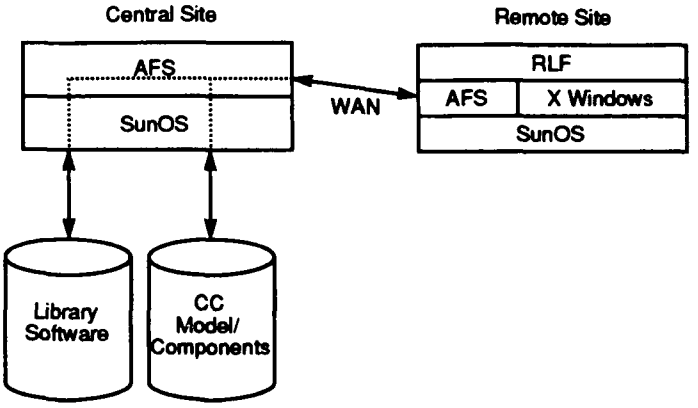

Figure 3-4. CC Library Fully Distributed Via AFS

The disadvantage to having all software distributed via AFS is the potential for reduced performance. On execution of the library software, the software must be cached into the remote access site's file cache. This initial caching operation is noticeably slow. However, after the software is cached, future access will typically not induce the same degradation in performance until the software is updated. Each time the library software is updated on the central site, the next access will again cause the AFS cache to be reloaded.

In using the model of full distribution via AFS, responsiveness of the initial connection depends on the throughput of the network. Once the AFS cache contains a copy of the library software, the throughput is much less of a factor because nearly all references are to the local software and model and not to the actual components stored at the central site. On the first retrieval of an component, throughput is again a factor. Successive accesses to the same component do not incur additional network traffic. Throughput consistency becomes a factor if the AFS cache is so small that the library software is frequently purged. This is resolved simply by creating a larger AFS cache. Throughput consistency is also a factor when retrieving components.

3.2.2 Partial distribution via AFS

It is expected that the library software will reside on selected remote site workstations when:

- a very slow link between the central and remote sites exists, or
- the AFS cache tends to be purged, forcing the library software to be loaded into the cache too frequently.

Partial distribution (Figure 3-5) has the clear advantage that start-up times are consistently fast. The disadvantage is that each of the remote sites' software will be managed separately from that at the central site. Update of the software could be automated, reducing the problems typically encountered in this situation (i.e., version control). This approach also requires a large amount of storage space to be allocated independently of the AFS cache for the storage of the software.

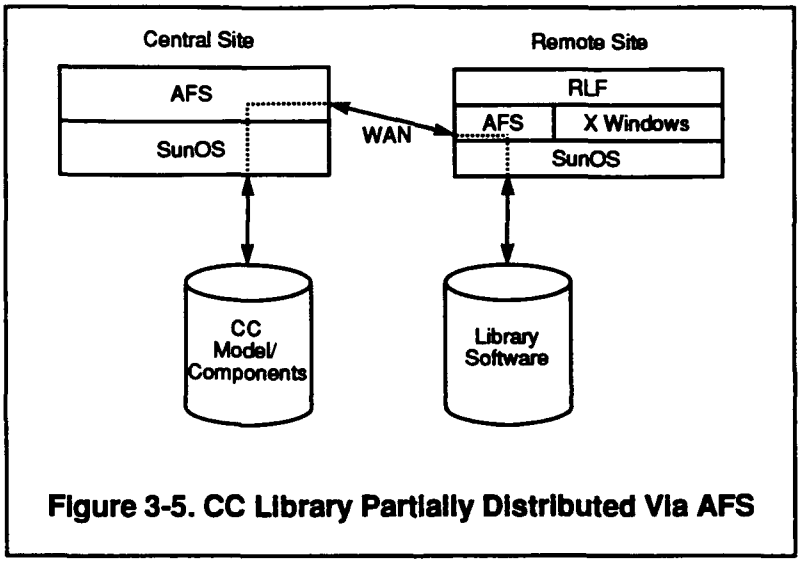

Figure 3-5. CC Library Partially Distributed Via AFS

Even with the software stored at the remote sites, it is anticipated that the model and the CC components will be accessed via AFS and will be maintained at the central site. While the model will need to be accessed each time the library is started, the model files themselves are rather small, so network access to them is quick. The library model and frequently accessed files will remain active in the local workstation's cache thus further reducing access time across the WAN.

When only partial AFS distribution is used, then network responsiveness is far less of an issue, since only retrieval of the model and library components is affected. The library software is always available at the remote site. In this case, low throughput has a mild impact on start-up performance. Component retrieval performance is the same as with full distribution.

3.2.3 Distribution via the X window system

Human-machine interface to the library is via the X Window System (or simply X). X is also a naturally distributed system, with the ability to operate graphical displays that are remote from the central site (see Figure 3-6).

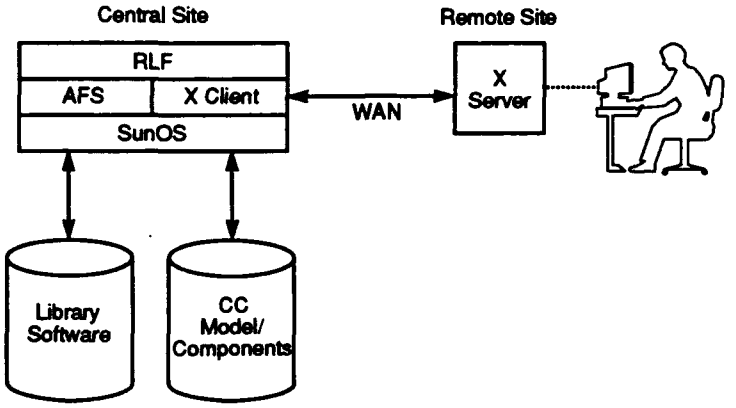

Figure 3-6. CC Library Distributed Via X

In this scenario, the user works from a remote X-capable interface (either a workstation or an X terminal) running X, and runs the library software on the central site while displaying the user interface on the user's workstation. Access to the library software and model is very fast as there is no transfer of files across the network. However, each interaction with the system involves network access. Therefore, one could encounter network access disruption or slow response time due to heavy network traffic.

The advantage to this mechanism is that it does not require a workstation for remote access, only a lower-cost X terminal. This solution also adopts the principal advantage of using AFS for complete distribution services, in that the central site still manages the complete software configuration, with no burdens adopted by the remote site.

Disadvantages include the instability of responsiveness and the need for the user to have access to the central site beyond access to the file system.

3.3 Telecommunications

CARDS utilizes network technologies and protocols that are compatible with DoD and government-wide telecommunications policies. CARDS network connections will continue to be established incrementally. Initial nodes are located at organizations serving as beta sites for library demonstration and testing. The CC library will continue this initial network of active reuse organizations to assess and evaluate network effectiveness and design alternatives.

All workstations at the CC library site and at each remote site should be interconnected via local-area networks (LAN), thus using a single connection to a wide-area network (WAN). For the WAN connecting the remote sites with the local site, there are many considerations. The three most critical items are:

1. throughput/consistency

2. direct connection
3. connection through a WAN (e.g., Internet).

Since effective distribution of information and components is critical to the success of the library, the CC library provides a fast and efficient means of transferring electronic information between the central and remote sites.

Individual site connection requirements are dependent upon the amount of data to be transmitted. Software required for network connection is dependent upon each site's preexisting computer and network resources. For example, the combination of data rate, local hardware and local software at a given site may call for either leasing a dedicated line or upgrading the network connection.

Development strategies that reduce the level of network traffic required for user interaction with the CC library will continue to be examined. Such strategies may include: development of library interface modules which reside on individual users' platforms (hence removing network traffic pertaining to screen display and refresh transmission); or distribution of library-encoded domain model/components to reside on individual user's systems.

3.3.1 Wide-area networks

Wide-area networks, such as Internet, provide an effective access mechanism to allow users to review and extract components in a timely manner, without direct support by the library personnel. As such, the WAN implementation strategy addressed in detail in the CARDS Library Operations Policies and Procedures (LOPP) [13] consists of:

- equipment required to connect a site to the network
- how quickly a site can be connected to the network
- network restrictions and limitations
- cost of connecting a site to the network
- local communication requirements
- communication software for the network.

Access to the CC library via Internet adds significant advantages to its usefulness. Remote access sites that already have Internet access will only need to install the X Window System and AFS to access the library. This allows new sites to be connected for experimental use with minimal overhead and delay. Because AFS enables security to be established in very flexible ways, access can be sufficiently free to allow any AFS-capable site to use the library, or restrictive enough to prevent access to everyone except those with specific authorization.

3.3.2 Direct connection

If a direct connection is chosen, allowing other Internet traffic to be passed across the link becomes an issue. If other-site Internet traffic is allowed, large, sporadic transfers across the link

could significantly reduce the consistency with which the network responds. With the software architecture chosen, this effect on responsiveness is limited to specific, predictable points during user access to the archive, which minimizes its seriousness. The direct links from the CC library are restricted so that other Internet traffic is eliminated.

3.4 Security

With the popularity and, in some cases, the necessity of open systems and wide area networks and the corresponding increase in security breaches in such systems, computer security is an issue that must be addressed. Security incidents range from manual attacks (by both "hackers" and insiders) to automated attacks (such as worms). Any incident which has the capacity to inhibit the operation of the library or system can be identified as a security problem, without regard to the source (authorized or unauthorized personnel) or mechanism (accident or deliberate).

Computer security is often associated with hackers breaking into systems and viruses disrupting service, but it encompasses much more than this. Threats may also be categorized as natural (e.g., earthquake), structural (e.g., flaws in the physical environment), and unintentional human errors. Any event that has the potential of resulting in disclosure of information, modification or loss of data, denial of service, or fraud, waste, and abuse should be considered a potential security threat.

Development of a secure system is an iterative process involving four steps:

1. Development of a security policy
2. Determination of the scope of the security policy
3. Preparation of a risk analysis
4. Implementation of countermeasures

Iterations are required due to the ever-changing world of computer hardware and software, and the corresponding changes that are required of systems that are based on the hardware and software. As a system (such as CARDS) matures, its scope or policies may change, requiring that the security policy be reviewed and updated correspondingly.

3.4.1 Security Concept of Operations

The Security Concept of Operations, as referenced in the LOPP [13], addresses the four steps presented above and their relationship to CARDS. Potential threats to the CARDS system's application software, operating system software, and library components are presented. The risk associated with each threat is analyzed and countermeasures suitable to reducing or eliminating the potential for threat are presented. The following sections present an overview of the concepts involved in security. The information presented here is a small subset of the security issues pertaining to the CARDS library infrastructure structure.

3.4.2 Security policy

The first step in developing a secure system is preparation of a security policy, derived from the library concept of operations. Its purpose is to state what must be protected and from whom it is being protected. CARDS current security policy is being included in the LOPP with the next update due in November 1992. The CARDS security policy is outlined below:

- Limit library access to authorized users
- Limit use of the library to that of its intended purpose
- Ensure continued service of the reuse library
- Ensure the integrity of new library components
- Protect the integrity of existing library components
- Protect the licensing and distribution of COTS
- Protect the distribution of COTS

3.4.3 Scope of CARDS security

Initial analysis and implementation of CARDS security encompasses only a portion of the overall security picture. Those areas of concern are:

- Administrative security
- Computer security
- File security
- Communications security

Only electronic threats (viruses, unauthorized users, etc.) will be considered. Physical threats, such as fire, will be considered at a later time.

The Security Concept of Operations covers only those components to be stored in the library as of Phase II:

- Nonclassified
- COTS
- GOTS
- Public domain

Due to legal and financial repercussions, licensing of COTS must be addressed. The library must be concerned with, and protect against, proprietary components being taken without the proper authorization. Additionally, distribution levels of government materials must also be considered.

To keep the analysis manageable, the scope is initially limited by making a number of assumptions, including:

- The focus of the analysis is the CARDS CC library.
- There is a closed security environment (developers and configuration management controls are trusted, i.e., malicious actions from these sources are not considered).
- Users are either government or government contractors.
- Personal contents of accounts and work areas are not covered.

3.4.4 Risk analysis

Risk analysis is a process that addresses risks to a computer system and its components over the entire life-cycle for that system. Risk is a combination of threats to the system and the system vulnerabilities. Identification of those threats and vulnerabilities (collectively referred to as threats) is only the first step in the multi-phased process of risk analysis [8]:

- Threat identification
- Threat evaluation
- Countermeasure identification
- Threat re-evaluation

The approach taken in the CARDS risk analysis is a derivation of the methodologies presented in the Department of the Air Force's policies on computer security [7] and risk analysis [8].

3.4.4.1 Threat Identification

A truly secure system has countermeasures in place for all possible threats, asserting the importance of identifying those threats. The process of identifying and documenting those threats can be tedious and time consuming.

Encompassing a wide range of technologies, each with its own abundance of potential security threats, CARDS poses special security problems. Attempting to identify the risks to the CARDS library is, understandably, an formidable task. Viewing CARDS not as a system itself, but a collection of smaller subsystems (each with its own security problems), provides a means of making the task more manageable while keeping the integrity of the analysis. The following CARDS security areas were identified:

- Hardware
- System administration
- Run-time software (includes the operating system and other software)
- RLF
- Library components
- AFS
- Accounts (both user and developer)

3.4.4.2 Threat evaluation

Determining threats and countermeasures is not sufficient to insure that security is properly handled. Factors such as the likelihood, impact, or outcome of a potential security threat and the cost of appropriate countermeasures all contribute to the decision of whether or not to counter a potential security threat. Additionally, the security policy is checked for impact and updated locally as required every time a potential security threat is identified.

3.4.5 Countermeasures

After identifying threats, suitable countermeasures should be determined to reduce the likelihood and impact of an attack. Countermeasures will be implemented for each threat whose potential risk warrants the cost of implementation. The decision to implement a countermeasure depends on two factors: implementation vs. restoration costs and the effectiveness of the countermeasure.

An important part of identifying and evaluating countermeasures is the implementation cost. Many factors such as manpower, money, and system downtime must be considered to derive a true cost evaluation.

Whether a countermeasure can be justified for implementation largely depends on the cost of restoring the system to its state before the attack, if possible. If the cost of countering a potential risk outweighs the restoration cost, then it may not be viable to implement the countermeasure. This mentality, though, is only partially justifiable and could cause problems in the future if the frequency of a threat is not also taken into consideration. A threat which occurs, or has the potential to occur, frequently may warrant implementing a countermeasure, without regard to the cost.

4 Library Interoperability

STARS envisions that:

> *"reuse in the future will occur in the context of a distributed network of heterogeneous domain-specific libraries. Each library will likely focus narrowly on one or a small set of vertical or horizontal domains, since libraries emphasizing relatively narrow domains are more likely to yield high impact reuse through greater depth of focus and better control of variability. However, this proliferation of domain-specific libraries will promote library heterogeneity, since the libraries will utilize distinct data models designed specifically to capture the characteristics of the their respective domains."* [20]

In this distributed, heterogeneous library context, one of the key challenges is the establishment of mechanisms to allow users at a given host to locate, inspect, and reuse components within the entire library network, thus maintaining the uniqueness and enhancing the effectiveness of each library.

A long-term goal of CARDS is to provide a consistent method of access to and from other relevant libraries, enabling reusable components from these other libraries to become integral parts of the overall CC library and other yet to be developed libraries. CARDS will be part of an organizational framework that will allow for component utilization across physical library boundaries. It will strive to adhere to emerging industry standards for library interoperability (e.g., Asset Library Open Architecture Framework (*ALOAF*) or Reuse library Interoperability Group (*RIG*)).

To facilitate initial CC library interoperability, CARDS is investigating point solutions for interoperating with ASSET incorporating the existing and evolving standards being developed by ALOAF and the RIG. The CARDS/ASSET plans for interoperability call for implementation of the two scenarios during FY92. The lessons learned that CARDS and ASSET derive from this initial implementation of component exchange will be used to begin a similar process with the Defense Information Systems Agency's (DISA) Defense Software Repository System (DSRS).

In the process of formulating the plans and technical infrastructure for interoperability, business issues and technical issues must be addressed. Figure 4-1 depicts a general model for interoperability incorporating material produced by the RIG and ALOAF interoperability efforts. This model is generic in nature and can be utilized for formulating detailed plans for interoperability between various libraries.

4.1 A reference model for interoperability

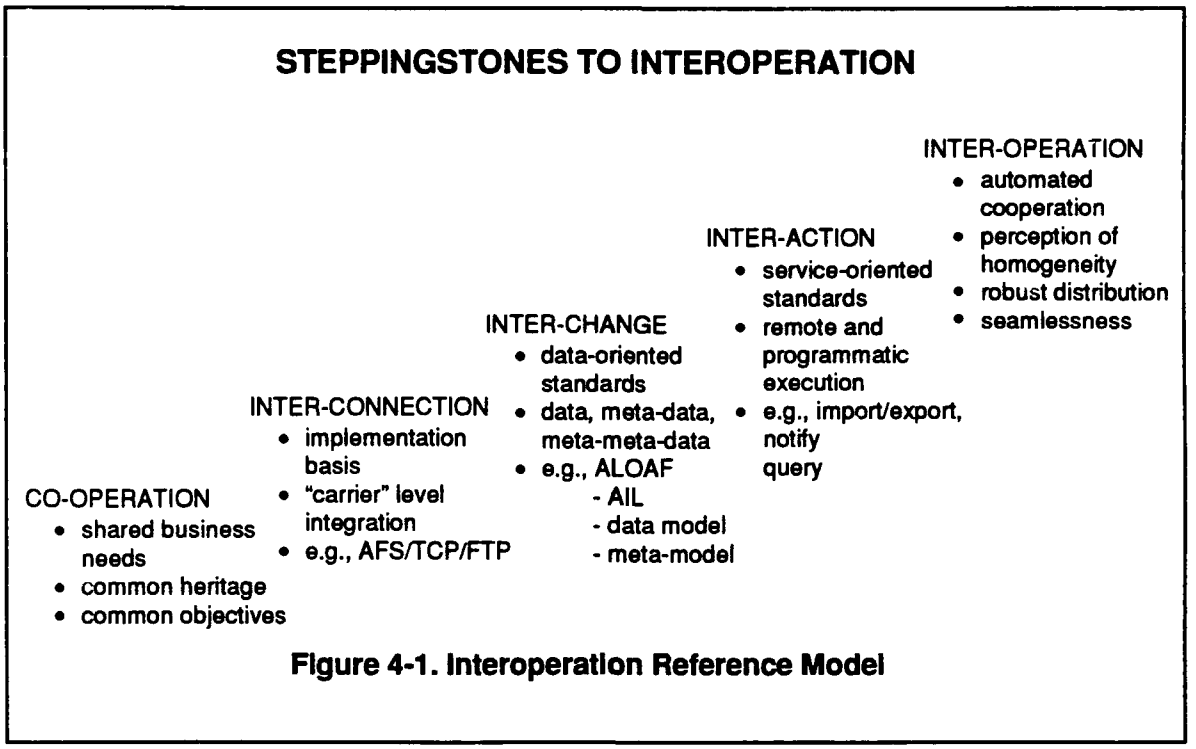

Figure 4-1. Interoperation Reference Model

- Co-operation: The impact of interoperability on the unique business needs and policies must be measured against the positive gains of interoperating. *Memorandums of understanding* between libraries stating the common objectives, goals and agreements builds the foundation for interoperability.
- Inter-connection: In order to achieve interoperability, an agreement must include the underlying mechanisms for connecting libraries that form the basic technical infrastructure. Common protocols for data exchange build this technical foundation. These common protocols can differ for each library connection.
- Inter-change: The interconnecting libraries incrementally establish data-oriented standards supporting data, data models and meta-models. An understanding of the other libraries unique models can help facilitate interoperability. The creation of agreed upon data models will facilitate inter-change.
- Inter-action: As library interoperability progresses incrementally through the reference, model service-oriented standards, remote and programmatic execution, can be described and implemented. The import/export, notification, query services and other specialization services that enhance library interoperability and begin to eliminate the user intervention or librarian to librarian notification are developed.
- Inter-operation: The final and optimal level of interoperations occurs with automation of the previous services, especially the librarian manual and user intervention processes, so that libraries are perceived to be homogeneous. (seamless Interoperability)

4.2 Scenarios

Scenarios for interoperation are a vehicle that can be utilized to understand the impact on internal library policies and the technical issues confronting each library for each step in the reference model. Several features that constitute a scenario for interoperability include:

- Automation (interactive vs. non-interactive) This describes whether the implementation is completely automated, or whether manual intervention on the part of a librarian is required to render the service.
- Actor (surrogate retrieval vs. user-retrieval) All asset interchange operations are services provided to a library user. Actor defines who performs the operation.
- Asset Protection (public vs. private) This describes whether the retrieved asset has been restricted in some way.
- Retrieval (direct vs. indirect) This describes whether an asset is retrieved directly from the library by either the interconnecting library or the user.
- Transparency (transparent vs. non-transparent) This refers to whether the end-user was made aware that an asset was retrieved from a cooperating library system.

Combinations of the above features can produce varied and multiple scenarios. For example, the following general retrieval scenarios can be used to define the user-to-library and library-to-library interaction. Illustrated below are three retrieval scenarios (direct retrieval, indirect retrieval and surrogate retrievals) that show different combinations of the above features.

4.2.1 Direct retrieval

With direct retrieval the user is interacting directly with the library in which the component resides. As figure 4-2 depicts, the interaction is user-to-library-to-user. Only one library is involved in this scenario and does not include interconnection with another library. This scenario illustrates the

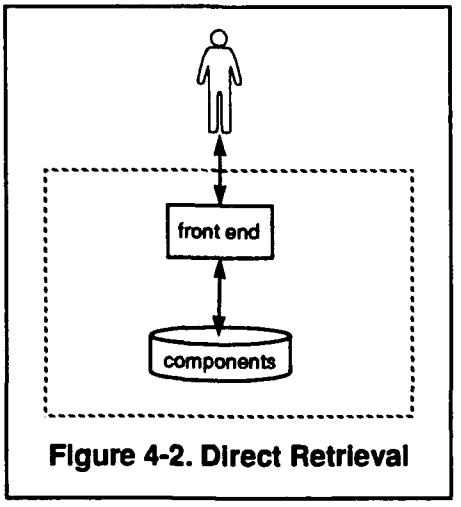

Figure 4-2. Direct Retrieval

automated (interactive), user-retrieval, transparent retrieval features.

4.2.2 Indirect retrieval

With indirect retrieval the user is able to retrieve components from another library. Depicted in Figure 4-3 are two methods for this type of user interaction:

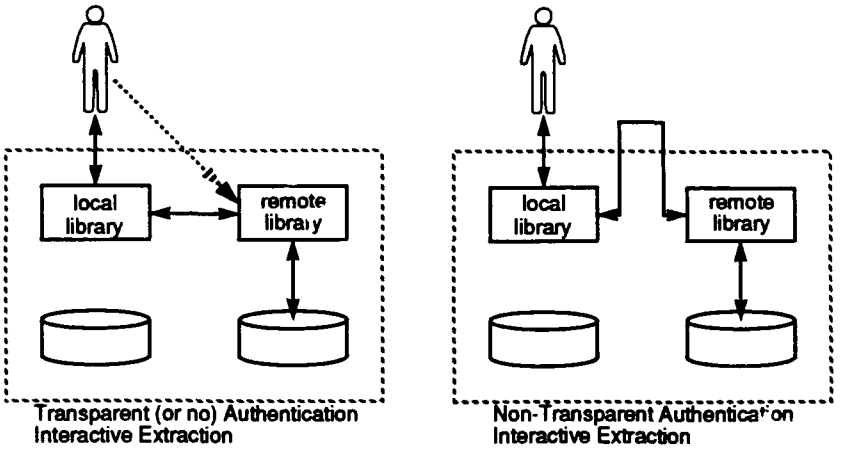

Figure 4-3. Indirect Retrieval

- The transparent, interactive extraction depicted in figure 4-4 depicts the libraries creating a mechanism by which the user's library can retrieve an asset for the user. The mechanisms in this instance can be automated or non-automated. The features illustrated are interactive, transparent, indirect retrieval.
- The Non-transparent, interactive extraction depicted in figure 4-3 requires that the user identify (authenticate) themselves to the remote library as is the case in private asset protection. The features illustrated are non-transparent, interactive, indirect retrieval.

4.2.3 Surrogate retrieval

The user's library retrieves a component from the remote library for the user. Between the two libraries there will exists an agreement that each library can be a surrogate user of the other library. Depending on the implementation the retrieval can occur interactively or not (See Figure 4-4). The scenarios depicted in figure 4-4 illustrate two combinations of features 1) surrogate, interactive, transparent, indirect retrieval; 2) surrogate, non-interactive, non-transparent, indirect retrieval.

4.3 CARDS/ASSET Interoperability Plan

The CARDS/ASSET interoperability plan enumerates the conditions and technical implementation of interconnecting the two libraries. Three scenarios have been developed for CARDS/ASSET interoperability to be demonstrated by 8 October 1992. The basic underlying technical framework for this demonstration includes an internet connection, SMTP (Simple Mail Transfer Protocol), AFS and the development of an ASSET index and a CARDS index.SMTP will

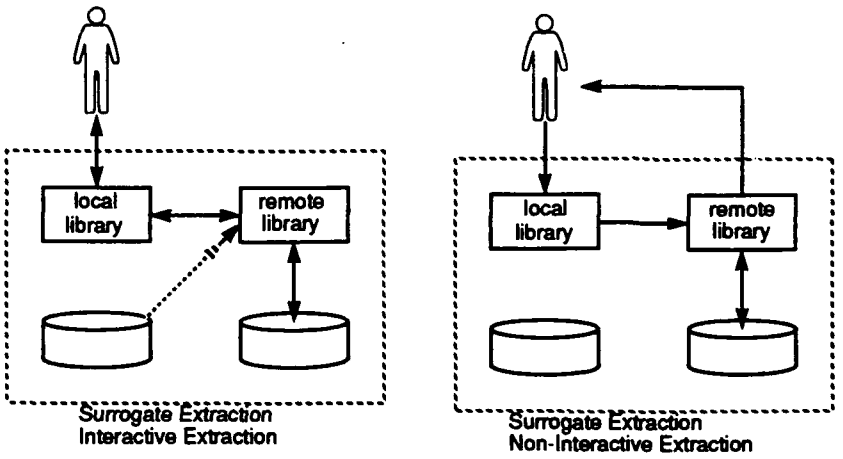

Figure 4-4. Surrogate Retrieval

provide the E-mail facility used in the notification process. AFS, a distributed file system, will be utilized as the asset exchange facility. Indexes for each libraries assets were developed to facilitate asset exchange. The indexes contain essential information each library needs inorder to understand and incorporate the asset information into their library. The CARDS/ASSET Interoperability Plan contains the detailed information on the three scenarios being implemented, the ASSET/CARDS index, the approach, the schedule and milestones for ASSET/CARDS interoperability.

Appendix A Glossary

AdaKNET - A semantic network modeling subsystem written in Ada. It provides the heart of the Reuse Library Framework's library model.

AdaTAU - A rule-based inferencing subsystem written in Ada. It supports the AdaKNET semantic network in the Reuse Library Framework's library model.

AFS - A distributed, wide-area network file system.

ALOAF - Asset Library Open Architecture Framework. The conceptual structure that supports seamless interchange and interoperability among networked, distributed heterogeneous component libraries by defining a service model; protocols supporting that model; Ada package specifications for the protocols; and a component exchange common data model, semantics, and formats.

ASSET - Asset Source for Software Engineering Technology. A reuse library containing a broad spectrum of components.

application domain - The knowledge and concepts which pertain to a particular computer application.

architecture-level integration - Combining architecture level components to create a system architecture or domain architecture.

architecture model - A model that represents the interrelationships between system elements and sets a foundation for later requirements analysis and design steps.

attribute - A characteristic of an object or relationship. Each attribute has a name and a value.

authentication - The process of establishing that someone or something is who they say they are.

browse - Surveying the reusable component descriptions in a library to determine whether the component is applicable to the current application.

CLIPS - C Language Integrated Production System.

COTS - Commercial Off-The-Shelf. Commercially available software.

classification - A mapping of a collection of objects to a taxonomy; the process of determining such a mapping.

command center (CC) - A facility from which a commander and his/her representatives direct operations and control forces. It is organized to gather, process, analyze, display and disseminate planning and operational data and to perform other related tasks.

component - A set of reusable resources that are related by virtue of being the inputs to various stages of the software life cycle, including requirements, design, code, test cases,

documentation, etc. Components are the fundamental elements in a reusable software library.

component acquisition - The process of obtaining components appropriate for reuse to be included in a library.

component qualification - The process of determining that a potential component is appropriate to the library and meets all quality requirements. Evaluation takes places against domain criteria.

concept - An atomic part of the AdaKNET knowledge representation scheme, representing an idea or thing.

context - The circumstances, situation, or environment in which a particular system exists.

context model - To model the scope of the domain as it exists within a larger domain denoting inputs and outputs.

DISA - Defense Information Systems Agency.

DSRS - Defense Software Repository System.

data model - A logical representation of a collection of data elements and the association among those data elements.

domain - An area of activity or knowledge containing applications which share a set of common capabilities and data.

domain analysis - The process of identifying, collecting, organizing, analyzing, and representing the relevant information in a domain based on the study of existing systems and their development histories, knowledge captured from domain experts, underlying theory, and emerging technology within the domain.

domain architecture - High-level paradigms and constraints characterizing the commonality and variances of the interactions and relationships between applications within a domain.

domain criteria - Specifications a component must adhere to in order to obtain acceptability in the domain. See component qualification.

domain engineering - An encompassing process which includes domain analysis and the subsequent construction of components, methods, tools, and supporting documentation that address the problems of system/subsystem development through the application of the knowledge in the domain model and software architecture.

domain expert - An individual who is knowledgeable in a domain.

domain-level integration - The process of using and evolving domain and application components in the creation of requirements, architectures and implementations (domain and

application).

domain model - A definition of the functions, objects, data, and relationships in a domain, consisting of a concise representation of the commonalities and differences of the problems of the domain and their solutions.

domain modeling - The process of encoding knowledge about a domain into a formalism.

domain-specific library - A library whose components are bound by a specific domain.

domain-specific reuse - Reusing components in a specific domain (through the use of a domain-specific library) to build an instance of an application in that domain.

ERA model - Entity-Relationship-Attribute model. Models data objects and their relationships using a graphical notation.

encode - To convert a domain model into a library model.

GOTS - Government Off-The-Shelf. Software developed for and owned by the government.

generic architecture - High-level paradigms and constraints that characterizing the commonality and variances of the interactions and relationships between the various components in a system.

generic command center architecture - The fundamental generic architecture that underlies command center applications.

graphical browser - A graphical presentation of the domain model and interrelations between components. Through the graphical browser, components may be browsed, viewed, and extracted. It also provides an inferencing mechanism to aid in prototyping and selecting the correct components.

horizontal domain - The knowledge and concepts that pertain to a particular functionality of a set of software components that can be utilized across more than one application domain.

implementation-level integration - Combining components in order to implement a system.

infrastructure - The basic underlying framework or features.

interoperability - The ability of two or more systems to exchange information and to mutually use the information that has been exchanged.

knowledge blueprint - A flexible plan to transition knowledge to the community.

knowledge representation - Codification of domain knowledge.

library - A collection of components that are cataloged according to a common classification scheme and a set of applications that provide a mechanism to browse and retrieve components.

library applications - Services provided to the library user.

library-centered domain-specific reuse - Reusing components in a specific domain to build an instance of an application in that domain utilizing a domain specific reuse library.

library model - A model that represents the domain components and the relationships between them.

life-cycle - All the activities (e.g., design, code, and test) a component is subjected to from its inception until it is no longer useful. A life cycle may be modeled in terms of phases, which are often characterizations of activities by their purpose or function such as design, code, or test.

life-cycle artifact - A product of the software engineering process (i.e., a component).

memorandum of understanding - An agreement stating terms of cooperation between two entities.

model - A representation of a real-world process, device, or concept.

modeling - The process of creating a model.

PRISM - Portable Reusable Integrated Software Modules.

prototyping - The practice of building a first or original model (sometime scaled down, but accurate) of a system to verify the operational process prior to building a final system.

RLF - Reuse Library Framework. Provides a framework for building domain-specific libraries.

RIG - Reuse library Interoperability Group. An industry/government group working to form a consensus of basic services for interoperability.

rapid prototyping - The process of using a library mechanism to quickly prototype a system.

relationships - The connections between entities, objects, or components.

repository - The mechanism for defining, storing, and managing all information concerning an enterprise and its software systems - logical data and process models, physical definitions and code, and organization models and business rules.

retrieval - The process of obtaining a component from a library such that it may be used in the development process.

reusable component - A component (including requirements, designs, code, test data, specifications, documentation, expertise, etc.) designed and implemented for the specific purpose of being reused.

reuse - The application of existing solutions to the problems of system development. Reuse involves transfer of expertise encoded in software-related work products. The simplest form of reuse from software work products is the use of subroutine/subprogram libraries

for string manipulations or mathematical calculations.

reuse library - A library specifically designed, built, and maintained to house reusable components.

reuser - One who implements a system through the process of reuse.

SADT - Structured Analysis and Design Techniques. A system analysis and design technique used for system definition, software requirements analysis, and system software design.

STARS - Software Technology for Adaptable, Reliable Software.

semantic network - A graphical knowledge representation method composed of nodes linked to each other.

software architecture - High-level paradigms and constraints characterizing the structure of operations and objects, their interfaces, and control to support the implementation of applications in a domain. Includes the description of each software component's functionality, name, parameters and their types, and a description of the component's interrelationships.

specialization - The act of declaring that one concept represents a narrowing of the idea represented by another concept.

surrogate retrieval - A user's library retrieves a component from a remote library for the user.

system architecture - A model that represents the interrelationship between system elements and sets a foundation for later requirements analysis and design steps.

system composition - The automatic configuration of a prototype system based on hardware and software requirements.

system engineering - A process encompassing requirements gathering at the system level with a small amount of top-level design and analysis.

taxonomy - The theory, principles, and process of categorizing entities in established categories.

vertical domain - The knowledge and concepts that pertain to a particular application domain.

Appendix B References

1. AdaKNET User's Manual, 1991,Informal Technical Report, STARS, Paramax

2. AdaTAU User's Manual, 1991, Informal Technical Report, STARS, Paramax

3. AFS System Administrator's Guide, 1991,Transarc Corp. Pittsburgh, PA

4. Arango, Guillermo F., 1988, Domain Engineering for Software Reuse, Ph.D. Thesis, University of California at Irvine.

5. Brachman R., 1978, A structural Paradigm For Representing Knowledge, Bolt Beranek and Newman, Inc.

6. Cohen, Sholum, 1992, Software Reuse Technology: Feature-Oriented Domain Analysis, SEI Tutorial Slides.

7. Department of the Air Force, AFR 205-16, Computer Security Policy, April 1989.

8. Department of the Air Force, AFSSM 5018, Risk Analysis Guide, November 1991.

9. Devonbu P., R.J. Brachman, P.G. Selfridge, and B.W. Ballaard, 1990, LaSSIE: A Knowledge-Based Software Information System, Proceedings of the 12th International Conference on Software Engineering, pages 249-261.

10. D'Ippolito, Richard S., 1989, Using Models in Software Engineering, Proceedings of TRI-Ada'89, pages 256-265 ACM, New York, NY.

11. DISA Command Center Design Handbook, 1991.

12. Goguen, Joseph A., 1986, Reusing and Interconnecting Software Components, IEEE Computer, Volume 19, Number 2, pages 16-28.

13. Library Operations Policies and Procedures for the Central Archive for Reusable Defense Software, Informal Technical Report, STARS, Paramax, 1992.

14. Lonardo, G.G., J.D. Wallin, 1992, PRISM Qualification Methodology Report, ESC, Hanscom AFB, MA

15. Neighbors, James M., 1989, DRACO: A method for Engineering Reusable Software Systems, Software Reusability, Volume 1, ACM Press, pages 295-320

16. Pietro-Diaz, Reuben, 1991, Reuse Library Process Model, STARS, IBM.

17. Pietro-Diaz, Reuben and Peter Freeman "Classifying Software for Reusability," *IEEE Software*, January 1987, 6-16.

18. Software Repository Report for the PRISM Program, 1992, ESD, Hanscom AFB, MA.

19. Wallnau, Kurt C., James J. Solderitsch, Mark A. Simos, Raymond C. McDowell, Keith A. Cassell, and David J. Campbell *Construction of Knowledge-Based Components and Applications in Ada.* Unisys - Paoli Research Center.

20. STARS Reuse Concept Volume 1 - Conceptual Framework for Reuse Process Version 1.0, 1992, STARS-TC-04040/001/00, IBM, Paramax, Boeing.